Etape 13 : Premiers RdV

Multithreading en JAVA

Synchronisation entre Thread : 1. premiers RdV

- En général on fait appel au multi-threading pour des traitements en parallèle mais ça n'exclut pas de devoir parfois réodonner l'exécution des thread sur mode séquentiel
- Une possibilité est offerte : démarrer un thread lorsqu'un autre se termine :

join() suspend l'exécution du *thread* courant jusqu'à la fin d'un autre thread

on peut donner des délais de garde (au delà on atetnd plus) :
join(long delai) : attend au maximum *delai* ms,
join(long milli, int nano) : attend au maximum *milli* ms + *nano* ns

Synchronisation entre Thread : 1. premiers RdV

- Comme toujours *join* ne s'applique pas un autre (thread) mais à "soi-même" en attendant la fin d'un autre :

```
public class MonThread extends Thread
{
    public void run()    {
            System.out.println("Début du long traitement.");
            Thread.sleep(3000);
            System.out.println("Fin du long traitement.");
            }

public static void main(String[] args)
    {
             MonThread t = new MonThread();
            t.start();
            System.out.println("Attente de la fin du thread t.");
            t.join();
            System.out.println("Le thread t a fini son exécution."); }  }
```

Synchronisation entre Thread : 1. premiers RdV

- Quand un thread est suspendu en attende de la fin d'un autre il se trouve dans l'état TiME-WAITING
- => il devient sensible à une interruption : il faut gérer l'exception

```
public static void main(String[] args) {
try {
    MonThread t = new MonThread();
    t.start();
    System.out.println("Attente de la fin du thread t.");
    t.join();
    System.out.println("Le thread t a fini son exécution.");
    }
catch (InterruptedException ie) { ie.printStackTrace(); }
```

Synchronisation entre Thread : 1. join

- *Le join ajoute une transition dans l'automate des états d'un Thread :*